Touring a Datacenter, Finishing up TCP, and if time, The Web

15-441 Fall 2017
Profs Peter Steenkiste & Justine Sherry

Thanks to Scott Shenker, Sylvia Ratnasamay, Peter Steenkiste, and Srini Seshan for slides.
Preamble to Web

• Professor Steenkiste is traveling most of the month

• Anonymous Q&A

• I went to a Facebook datacenter
  • Guest speaker to tell us about data centers at Microsoft on 11/17

•_finishing up TCP
More About Datacenters

• Dr. David Maltz will be visiting our class 11/17
  • Invite your friends
  • He is a CMU alumnus!
  • He runs most of networking for Microsoft Azure (ie, their public cloud datacenters).
Preamble to Lecture

• Professor Steenkiste is traveling most of the month

• Anonymous Q&A

• I went to a Facebook datacenter
  • Guest speaker to tell us about data centers at Microsoft on 11/17

• Finishing up TCP
Quick Review

dupACK = 3

cwnd > ssthresh

timeout

dupACK

timeout

dupACK

timeout

dupACK = 3

new ACK

dupACK

timeout

new ACK

dupACK
Quick Review

- **Slow Start**
  - $cwnd > ssthresh$ (Fast Recovery)
  - $\text{dupACK}$
  - $\text{new ACK}$
  - $\text{timeout}$

- **Fast Recovery**
  - $\text{dupACK}=3$

- **Congestion Avoidance**
  - $\text{new ACK}$
  - $\text{dupACK}$
  - $\text{timeout}$
  - $\text{dupACK}=3$
Quick Review

dupACK

cwnd > ssthresh

dupACK=3

dupACK

timeout

new ACK

dupACK

timeout

new ACK

dupACK=3

timeout

new ACK
Quick Review

**slow start**
- timeout
- dupACK
- new ACK
- dupACK=3

**fast recovery**
- timeout
- new ACK
- dupACK

**congestion avoid.**
- timeout
- cwnd > ssthresh
- new ACK
- dupACK
- dupACK=3

Note: The diagram illustrates the three states of TCP congestion control: slow start, congestion avoidance, and fast recovery. The transitions are triggered by timeouts, duplicate ACKs, and new ACKs.
What’s wrong with TCP?
Did we meet our goals?
Did we meet our goals?

Wanted a mechanism that:

- Uses network resources efficiently
- Prevents collapse
- Preserves fair network resource allocation
A Simple Model for TCP Throughput

\[ cwnd \]

Loss

\[ W_{\text{max}} \]

\[ \frac{W_{\text{max}}}{2} \]

\[ t \]
A Simple Model for TCP Throughput

cwnd

\[ W_{\text{max}} \]

\[ \frac{W_{\text{max}}}{2} \]

Loss

1

\( \text{RTT} \)
A Simple Model for TCP Throughput

A

\[ cwnd \]

\[ W_{\text{max}} \]

\[ \frac{W_{\text{max}}}{2} \]

\[ \text{RTT} \]
A Simple Model for TCP Throughput

\[ \text{Loss} \]

\[ \frac{1}{2} W_{\text{max}} \text{ RTTs between drops} \]
A Simple Model for TCP Throughput

\[ cwnd \]

\[ W_{\text{max}} \]

\[ \frac{W_{\text{max}}}{2} \]

\[ t \]

\[ \frac{1}{2} W_{\text{max}} \] RTTs between drops

Avg. \( \frac{3}{4} W_{\text{max}} \) packets per RTTs
A Simple Model for TCP Throughput

Packet drop rate, \( p = 1 / A \), where \( A = \frac{3}{8} W_{\text{max}}^2 \)

Throughput, \( B = \frac{A}{\left( \frac{W_{\text{max}}}{2} \right) RTT} = \sqrt{\frac{3}{2} \frac{1}{RTT \sqrt{p}}} \)
Did we meet our goals?

Wanted a mechanism that:

• **Uses network resources efficiently?**
• Prevents collapse
• Preserves fair network resource allocation
Problem #1: Can’t scale to large BDP networks

Delay-bandwidth product for 100ms delay
- 1.5Mbps: 18KB
- 10Mbps: 122KB
- 45Mbps: 549KB
- 100Mbps: 1.2MB
- 622Mbps: 7.4MB
- 1.2Gbps: 14.8MB

Why is this a problem?
- 10Mbps > max 16bit window

Scaling factor on advertised window
- Specifies how many bits window must be shifted to the left
- Scaling factor exchanged during connection setup
Problem #1: Can’t scale to large BDP networks

- “Large window” option (RFC 1323)
  - Negotiated by the hosts during connection establishment
  - Option 3 specifies the number of bits by which to shift the value in the 16 bit window field
  - Independently set for the two transmit directions
- The scaling factor specifies bit shift of the window field in the TCP header
  - Scaling value of 2 translates into a factor of 4
- Old TCP implementations will simply ignore the option
  - Definition of an option
Even if we had enough sequence numbers, TCP algorithm can’t easily keep the link saturated.

\[ BW = \frac{MSS}{RTT \times \sqrt{2p/3}} \]

- Suppose RTT = 100 ms, MSS = 1.5 KB
- T = 100 Gb/sec
- p=?
  - \( p \approx 2 \times 10^{-12} \)
- 1 drop every 6 petabits (17 hours).
Same problem, visually.

- A TCP connection with 1250-Byte packet size and 100ms RTT is running over a 10Gbps link (assuming no other connections, and no buffers at routers)
Enter: TCP Cubic

- Goal is to spend more time at the high end of the window value range
- Remember: 1.4 hours to reach Wmax on 10 Gbps link?
- Idea: make the additive increase adaptive depending on how close you are to presumed Wmax value
- Fast recovery using larger additive increase toward Wmax (cubic increase)
  - Slow change around Wmax
  - Fast search for a higher Wmax
TCP Cubic in One Slide
Did we meet our goals?

Wanted a mechanism that:

• Uses network resources efficiently?
• **Prevents collapse** — **YES**
• Preserves fair network resource allocation
Did we meet our goals?

Wanted a mechanism that:

• Uses network resources efficiently?
• Prevents collapse
• Preserves fair network resource allocation
Is it fair?
Is it fair?

Throughput = \( \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}} \)

- Flows get throughput inversely proportional to RTT
Is it fair?

Throughput = $\sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$

- Flows get throughput inversely proportional to RTT
- TCP unfair in the face of heterogeneous RTTs!
Did we meet our goals?

Wanted a mechanism that:
• Uses network resources efficiently?
• Prevents collapse
• Preserves fair network resource allocation

One thing we forgot: latency
Latency

- TCP detects loss when buffers fill up and drop packets
  - This means that packets incur high queueing delay in the network.
  - When queues are long, this makes observed RTT go up!
- Consider a 100GB transfer sharing a link with a few 100B transfers.
- Recent work from Google on new algorithm (called BBR) that doesn’t fill up queues.
Today’s Network: Lots of TCP Algorithms

• TCP Cubic is the standard in Linux. Windows uses another algorithm called “Compound TCP”

• TCP NewReno (which we learned in this class) remains the “standard” — and everyone uses it as the reference.

• UDP-based protocols use their own algorithms for congestion control to avoid interfering with TCP.
Preamble to Lecture

• Professor Steenkiste is traveling most of the month

• Anonymous Q&A

• I went to a Facebook datacenter
  • Guest speaker to tell us about data centers at Microsoft on 11/17

• Finishing up TCP
ONWARD TO THE WEB!
1945: Vannevar Bush

- “As we may think”, Atlantic Monthly, July, 1945.
- Describes the idea of a distributed hypertext system
- A “memex” that mimics the “web of trails” in our minds
Dec 9, 1968: “The Mother of All Demos”

First demonstration of Memex-inspired system

Working prototype with hypertext, linking, use of a mouse…

https://www.youtube.com/watch?v=74c8LntW7fo
Many other iterations before we got to the World Wide Web

- MINITEL in France. https://en.wikipedia.org/wiki/Minitel
- (Note that you don’t need to know any of this history for exams, this is just for the curious…)
1989: Tim Berners-Lee

1989: Tim Berners-Lee (CERN) writes internal proposal to develop a distributed hypertext system

- Connects “a web of notes with links”.
- Intended to help CERN physicists in large projects share and manage information

1990: TBL writes graphical browser for Next machines

1992-1994: NCSA/Mosaic/Netscape browser release
Internet Traffic History

- **PByte/month**
- **Year**

Graph showing the growth of Internet traffic over the years with categories for **All**, **Fixed**, and **Mobile**.
Hyper Text Transfer Protocol (HTTP)
Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
  - server is “always on” and “well known”
  - clients initiate contact to server
Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
  - server is “always on” and “well known”
  - clients initiate contact to server

- Synchronous request/reply protocol
  - Runs over TCP, Port 80
Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
  - server is “always on” and “well known”
  - clients initiate contact to server

- Synchronous request/reply protocol
  - Runs over TCP, Port 80

- Stateless
Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
  - server is “always on” and “well known”
  - clients initiate contact to server

- Synchronous request/reply protocol
  - Runs over TCP, Port 80

- Stateless

- ASCII format
Steps in HTTP Request/Response

Client

Server
Steps in HTTP Request/Response

Client

Establish connection

TCP Syn

TCP syn + ack

Server
Steps in HTTP Request/Response

Client

Establish connection

TCP Syn

TCP syn + ack

Client request

TCP ack + HTTP GET

Request response

Server
Steps in HTTP Request/Response

Client

Server

Establish connection

TCP Syn

TCP syn + ack

TCP ack + HTTP GET

Request response

Close connection
Client-to-Server Communication

- HTTP Request Message

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr

(blank line)
```
Client-to-Server Communication

- HTTP Request Message
  - Request line: method, resource, and protocol version

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)
```
Client-to-Server Communication

- HTTP Request Message
- Request line: method, resource, and protocol version

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)
```
Client-to-Server Communication

- HTTP Request Message
  - Request line: method, resource, and protocol version

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)
```
Client-to-Server Communication

- HTTP Request Message
  - Request line: method, resource, and protocol version

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)
```
Client-to-Server Communication

- HTTP Request Message
  - Request line: method, resource, and protocol version
  - Request headers: provide information or modify request
Client-to-Server Communication

- **HTTP Request Message**
  - Request line: method, resource, and protocol version
  - Request headers: provide information or modify request
  - Body: optional data (e.g., to “POST” data to the server)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
(blank line)
```
Server-to-Client Communication

- HTTP Response Message

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html

(data)
```

- status line
  - (protocol, status code, status phrase)
- header lines
  - HTTP/1.1 200 OK
  - Connection close
  - Date: Thu, 06 Aug 2006 12:00:15 GMT
  - Server: Apache/1.3.0 (Unix)
  - Last-Modified: Mon, 22 Jun 2006 ...
  - Content-Length: 6821
  - Content-Type: text/html
- data
  - (e.g., requested HTML file)

...
Server-to-Client Communication

- HTTP Response Message
- Status line: protocol version, status code, status phrase
Server-to-Client Communication

- HTTP Response Message
- Status line: protocol version, status code, status phrase

status line
(protocol, status code, status phrase)

header lines

Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html

(data line)

data data data data data data data ...

e.g., requested HTML file
Server-to-Client Communication

- HTTP Response Message
- Status line: protocol version, status code, status phrase

**status line**
(protocol, status code, status phrase)

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html

(data line)
Data data data data data data ...
Server-to-Client Communication

- HTTP Response Message
- Status line: protocol version, status code, status phrase

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html
(data)

(e.g., requested HTML file)
Server-to-Client Communication

- HTTP Response Message
  - Status line: protocol version, status code, status phrase
  - Response headers: provide information

status line
(protocol, status code, status phrase)

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html
(blank line)
data data data data data data data ...

deleted message
(e.g., requested HTML file)
Server-to-Client Communication

- HTTP Response Message
  - Status line: protocol version, status code, status phrase
  - Response headers: provide information
  - Body: optional data

status line
(protocol, status code, status phrase)

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html
(data)
data data data data data data data ...

header lines

(data)
e.g., requested HTML file
HTTP is *Stateless*
HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state
HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state

- **Good:** Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn’t matter
HTTP is *Stateless*

- Each request-response treated independently
  - Servers *not* required to retain state

- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn’t matter

- **Bad**: Some applications *need* persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.*, Shopping cart, user profiles, usage tracking, …
State in a Stateless Protocol:

Cookies
State in a Stateless Protocol:

Cookies

- **Client-side** state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
State in a Stateless Protocol:

Cookies

- **Client-side state maintenance**
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
State in a Stateless Protocol:

Cookies

- **Client-side** state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server

```
Request

Set-Cookie: XYZ

Response
```
State in a Stateless Protocol:

Cookies

- **Client-side** state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
State in a Stateless Protocol:

Cookies

- *Client-side* state maintenance
  - Client stores small state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication
Performance Issues
Performance Goals
Performance Goals

● User
  ● fast downloads (not identical to low-latency commn.!)  
  ● high availability
Performance Goals

- **User**
  - fast downloads (not identical to low-latency communication!)
  - high availability

- **Content provider**
  - happy users (hence, above)
  - cost-effective infrastructure
Performance Goals

- **User**
  - fast downloads (not identical to low-latency communication!)
  - high availability

- **Content provider**
  - happy users (hence, above)
  - cost-effective infrastructure

- **Network (secondary)**
  - avoid overload
Solutions?

- User
  - fast downloads (not identical to low-latency commn.!)  
  - high availability

- Content provider
  - happy users (hence, above)
  - cost-effective infrastructure

- Network (secondary)
  - avoid overload
Solutions?

- User
  - fast downloads (not identical to low-latency commn.)!
  - high availability

- Content provider
  - happy users (hence, above)
  - cost-effective infrastructure

- Network (secondary)
  - avoid overload

Improve HTTP to compensate for TCP’s weak spots
Solutions?

- **User**
  - fast downloads (not identical to low-latency commun.)
  - high availability

- **Content provider**
  - happy users (hence, above)
  - cost-effective delivery infrastructure

- **Network (secondary)**
  - avoid overload

---

**Improve HTTP to compensate for TCP's weak spots**

**Caching and Replication**
Solutions?

- **User**
  - fast downloads (not identical to low-latency commun.)
  - high availability

- **Content provider**
  - happy users (hence, above)
  - cost-effective delivery infrastructure

- **Network (secondary)**
  - avoid overload

---

**Caching and Replication**

Improve HTTP to compensate for TCP’s weak spots
Solutions?

- User
  - fast downloads (not identical to low-latency commn.)
  - high availability

- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure

- Network (secondary)
  - avoid overload

Improving HTTP to compensate for TCP’s weak spots

Caching and Replication
Solutions?

- User
  - fast downloads (not identical to low-latency communication!)
  - high availability
- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure
- Network (secondary)
  - avoid overload

Caching and Replication

Improve HTTP to compensate for TCP’s weak spots
Solutions?

- User
  - fast downloads (not identical to low-latency commn.)
  - high availability
- Content provider
  - happy users (hence, above)
  - cost-effective delivery infrastructure
- Network (secondary)
  - avoid overload

Caching and Replication

Improve HTTP to compensate for TCP's weak spots
Solutions?

- User
  - fast downloads (not identical to low-latency commn.!) 
  - high availability

- Content provider
  - happy users (hence, above) 
  - cost-effective delivery infrastructure

- Network (secondary)
  - avoid overload

Caching and Replication

- Improve HTTP to compensate for TCP’s weak spots

Exploit economies of scale (Webhosting, CDNs, datacenters)
HTTP Performance
HTTP Performance

- Most Web pages have multiple objects
  - e.g., HTML file and a bunch of embedded images
HTTP Performance

• Most Web pages have multiple objects
  • e.g., HTML file and a bunch of embedded images

• How do you retrieve those objects (naively)?
  • *One item at a time*
HTTP Performance

• Most Web pages have multiple objects
  • e.g., HTML file and a bunch of embedded images

• How do you retrieve those objects (naively)?
  • *One item at a time*

• New TCP connection per (small) object!
Typical Workload (Web Pages)

- Multiple (typically small) objects per page
- File sizes
  - Heavy-tailed
    - Pareto distribution for tail
    - Lognormal for body of distribution
- Embedded references
  - Number of embedded objects also Pareto
    \[ \Pr(X>x) = \left(\frac{x}{x_m}\right)^{-k} \]
- This plays havoc with performance. Why?
- Solutions?
Typical Workload (Web Pages)

- Multiple (typically small) objects per page
- File sizes
  - Heavy-tailed
    - Pareto distribution for tail
    - Lognormal for body of distribution
- Embedded references
  - Number of embedded objects also Pareto
    \[ \text{Pr}(X>x) = \left(\frac{x}{xm}\right)^{-k} \]
- This plays havoc with performance. Why?
- Solutions?

- Lots of small objects versus TCP
  - 3-way handshake
  - Lots of slow starts
  - Extra connection state
Improving HTTP Performance:

**Concurrent Requests & Responses**
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses
Improving HTTP Performance: Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = 😊
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = 😊
- Content provider = 😞
Improving HTTP Performance:

**Concurrent Requests & Responses**

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = 😊
- Content provider = 😕
- Network = 😞 Why?
Improving HTTP Performance:

**Persistent Connections**
Improving HTTP Performance: **Persistent Connections**

- Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection
Improving HTTP Performance:  
**Persistent Connections**

- Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection

- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth
Improving HTTP Performance:

**Persistent Connections**

- Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection

- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth

- Default in HTTP/1.1
Improving HTTP Performance:

**Pipelined Requests & Responses**

- Batch requests and responses to reduce the number of packets
- Multiple requests can be contained in one TCP segment
Scorecard: Getting $n$ Small Objects
Scorecard: Getting \( n \) Small Objects

*Time dominated by latency*
Scorecard: Getting $n$ Small Objects

*Time dominated by latency*

- One-at-a-time: $\sim 2n$ RTT
Scorecard: Getting $n$ Small Objects

*Time dominated by latency*

- One-at-a-time: $\sim 2n$ RTT
- M concurrent: $\sim 2[n/m]$ RTT
Scorecard: Getting $n$ Small Objects

*Time dominated by latency*

- One-at-a-time: $\sim 2n$ RTT
- $M$ concurrent: $\sim 2[n/m]$ RTT
- Persistent: $\sim (n+1)$RTT
Scorecard: Getting $n$ Small Objects

*Time dominated by latency*

- One-at-a-time: $\sim 2n$ RTT
- M concurrent: $\sim 2[n/m]$ RTT
- Persistent: $\sim (n+1)$RTT
- Pipelined: $\sim 2$ RTT
Scorecard: Getting $n$ Small Objects

Time dominated by latency

- One-at-a-time: $\sim 2n$ RTT
- M concurrent: $\sim 2[n/m]$ RTT
- Persistent: $\sim (n+1)RTT$
- Pipelined: $\sim 2$ RTT
- Pipelined/Persistent: $\sim 2$ RTT first time, RTT later
Scorecard: Getting $n$ Large Objects
Scorecard: Getting $n$ Large Objects

Time dominated by bandwidth
Scorecard: Getting $n$ Large Objects

*Time dominated by bandwidth*

- One-at-a-time: $\sim nF/B$
Scorecard: Getting \( n \) Large Objects

*Time dominated by bandwidth*

- One-at-a-time: \(~ nF/B\)
- \( M \) concurrent: \(~ [n/m] F/B\)
  - assuming shared with large population of users
  - and each TCP connection gets the same bandwidth
Scorecard: Getting $n$ Large Objects

*Time dominated by bandwidth*

- One-at-a-time: $\sim nF/B$
- $M$ concurrent: $\sim \frac{n}{m} F/B$
  - assuming shared with large population of users
  - and each TCP connection gets the same bandwidth
- Pipelined and/or persistent: $\sim nF/B$
  - The only thing that helps is getting more bandwidth..
Improving HTTP Performance:

Caching

• Why does caching work?
  • Exploits *locality of reference*

• How well does caching work?
  • Very well, up to a limit
  • Large overlap in content
  • But many unique requests
Improving HTTP Performance:

Caching: How
Improving HTTP Performance:

Caching: How

• Modifier to GET requests:
  • `If-modified-since` – returns “not modified” if resource not modified since specified time
Improving HTTP Performance:

Caching: How

• Modifier to GET requests:
  • `If-modified-since` – returns “not modified” if resource not modified since specified time

```
GET /~ee122/fa13/ HTTP/1.1
Host: inst.eecs.berkeley.edu
User-Agent: Mozilla/4.03
If-modified-since: Sun, 27 Oct 2013 22:25:50 GMT
<CRLF>
```
Improving HTTP Performance:

Caching: How

• Modifier to GET requests:
  • `If-modified-since` – returns “not modified” if resource not modified since specified time

GET /~ee122/fa13/ HTTP/1.1
Host: inst.eecs.berkeley.edu
User-Agent: Mozilla/4.03
If-modified-since: Sun, 27 Oct 2013 22:25:50 GMT

• Client specifies “if-modified-since” time in request
  • Server compares this against “last modified” time of resource
  • Server returns “Not Modified” if resource has not changed
  • … or a “OK” with the latest version otherwise
Improving HTTP Performance:

Caching: How

• Modifier to GET requests:
  • If-modified-since – returns “not modified” if resource not modified since specified time

• Response header:
  • Expires – how long it’s safe to cache the resource
  • No-cache – ignore all caches; always get resource directly from server
Improving HTTP Performance:
Caching: Where?
Improving HTTP Performance:

Caching: Where?

- Options
  - Client
  - Forward proxies
  - Reverse proxies
  - Content Distribution Network
Improving HTTP Performance:

Caching: Where?

- Baseline: Many clients transfer same information
- Generate unnecessary server and network load
- Clients experience unnecessary latency
Improving HTTP Performance:

Caching with Reverse Proxies

- Cache documents close to **server**
  - decrease server load
- Typically done by content provider
Improving HTTP Performance:  
**Caching with Forward Proxies**

- Cache documents close to **clients**
  - reduce network traffic and decrease latency
- Typically done by ISPs or enterprises

![Diagram of network traffic with forward and reverse proxies](image-url)
Improving HTTP Performance:

Replication
Improving HTTP Performance: Replication

- Replicate popular Web site across many machines
  - Spreads load on servers
  - Places content closer to clients
  - Helps when content isn’t cacheable
Improving HTTP Performance:
Replication

- Replicate popular Web site across many machines
  - Spreads load on servers
  - Places content closer to clients
  - Helps when content isn’t cacheable

- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers
Improving HTTP Performance: Replication

- Replicate popular Web site across many machines
  - Spreads load on servers
  - Places content closer to clients
  - Helps when content isn’t cacheable

- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers

- Common solution:
  - DNS returns different addresses based on client’s geo location, server load, etc.
Improving HTTP Performance:
Content Distribution Networks
Improving HTTP Performance:
Content Distribution Networks

- Caching and replication as a service
Improving HTTP Performance:
Content Distribution Networks

- Caching and replication as a service
- Large-scale distributed storage infrastructure (usually) administered by one entity
  - *e.g.*, Akamai has servers in 20,000+ locations
Improving HTTP Performance:
Content Distribution Networks

- Caching and replication as a service
- Large-scale distributed storage infrastructure (usually) administered by one entity
  - *e.g.*, Akamai has servers in 20,000+ locations
- Combination of (pull) caching and (push) replication
  - **Pull**: Direct result of clients’ requests
  - **Push**: Expectation of high access rate
Improving HTTP Performance:
Content Distribution Networks

- Caching and replication as a service
- Large-scale distributed storage infrastructure (usually) administered by one entity
  - *e.g.*, Akamai has servers in 20,000+ locations
- Combination of (pull) caching and (push) replication
  - **Pull**: Direct result of clients’ requests
  - **Push**: Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*
Improving HTTP Performance:
CDN Example – Akamai
Improving HTTP Performance:

CDN Example – Akamai

- Akamai creates new domain names for each client
  - e.g., `a128.g.akamai.net` for `cnn.com`
Improving HTTP Performance:

CDN Example – Akamai

- Akamai creates new domain names for each client
  - e.g., a128.g.akamai.net for cnn.com

- The CDN’s DNS servers are authoritative for the new domains
Improving HTTP Performance:

CDN Example – Akamai

- Akamai creates new domain names for each client
  - e.g., a128.g.akamai.net for cnn.com

- The CDN’s DNS servers are authoritative for the new domains

- The client content provider modifies its content so that embedded URLs reference the new domains.
  - “Akamaize” content
  - e.g.: http://www.cnn.com/image-of-the-day.gif becomes http://a128.g.akamai.net/image-of-the-day.gif
Improving HTTP Performance:
CDN Example – Akamai

- Akamai creates new domain names for each client
  - e.g., `a128.g.akamai.net` for `cnn.com`
- The CDN’s DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - “Akamaize” content
  - e.g.: `http://www.cnn.com/image-of-the-day.gif` becomes `http://a128.g.akamai.net/image-of-the-day.gif`
- Requests now sent to CDN’s infrastructure…
Cost-Effective Content Delivery
Cost-Effective Content Delivery

- General theme: multiple sites hosted on shared physical infrastructure
  - efficiency of statistical multiplexing
  - economies of scale (volume pricing, etc.)
  - amortization of human operator costs
Cost-Effective Content Delivery

- General theme: multiple sites hosted on shared physical infrastructure
  - efficiency of statistical multiplexing
  - economies of scale (volume pricing, *etc.*)
  - amortization of human operator costs

- Examples:
  - Web hosting companies
  - CDNs
  - Cloud infrastructure
Thursday: Midterm Review.